

## Arquitetura de Computadores 2024/25

### TPC4

Este trabalho de casa consiste num exercício de programação a ser realizado em grupo de no máximo dois alunos. Pode esclarecer dúvidas gerais com colegas, mas a solução e a escrita do código devem ser estritamente realizadas pelos membros do grupo. Todas as resoluções serão comparadas de forma automática e os casos de plágio serão punidos de acordo com os regulamentos em vigor.

**Data/ Hora limite para a entrega: dia 07/06 (sábado) às 18:00.**

#### Atribuição dinâmica de memória e chamada ao SO *mmap*

Durante a execução de um programa, a obtenção de espaço na RAM é normalmente baseada nas funções de biblioteca do C, *malloc* e *free*. O espaço de memória usado para tal é chamado *heap*, sendo este um espaço contínuo que pode ser alterado com a chamada ao sistema *sbrk*. A figura seguinte resume o mapa de memória de um processo num sistema operativo como o Linux:

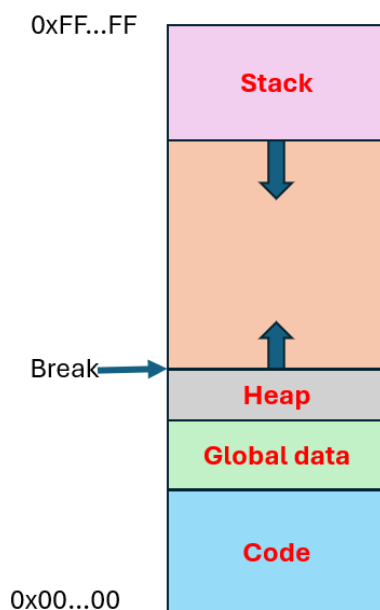


Figura 1 - Mapa de memória de um processo Linux

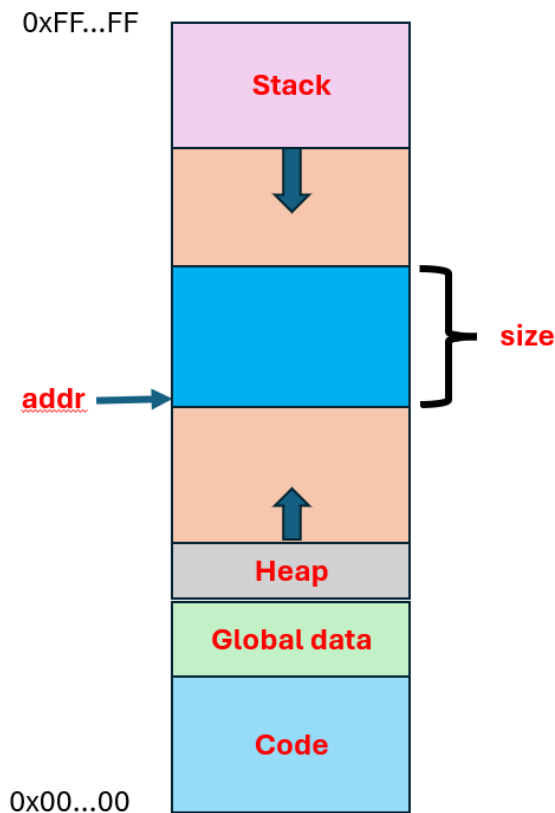
Um processo pode obter espaço na RAM através da chamada ao sistema *mmap*. Esta chamada ao sistema tem um conjunto de parâmetros complexo que pode ser consultado através do comando *man mmap*, mas no contexto deste trabalho, assume-se o conjunto de parâmetros exemplificado no programa C seguinte:

```
#include <sys/mman.h>
```

```
size_t size = ... ; // size must be a multiple of the size of a page (4096)
```

```
void *addr = mmap( 0, size, PROT_READ | PROT_WRITE , MAP_PRIVATE | MAP_ANONYMOUS, -1, 0 );
if( addr == (void *)-1 ){
    perror("mmap");
    return 1;
}
```

Se a chamada ao sistema tiver sucesso o mapa de memória passa a ser:



*Figura 2 - Mapa de memória após o uso da chamada ao sistema mmap*

O processo tem acesso à nova área através de instruções C tais como:

```
char *pt = addr;
pt[5] = pt[22] + 5; // reading and writing in the new allocated memory area
```

Neste trabalho, vamos implementar um conjunto de operações (API) que permitem o uso desta área de memória através de operações similares às operações *malloc* e *free* da biblioteca do C. A API está definida no ficheiro *myAlloc.h* cujo conteúdo é o seguinte:

```
typedef struct {
    void *base;           // beginning of managed area
    void *top;            // address of last byte allocated
    unsigned long int *limit; // last address in the area allocated by mmap
} heap;

typedef struct{
    unsigned int available;
    unsigned int size;
} block;

void allocate_init (heap *h, unsigned long int maximumSize);
```

```
void *allocate(heap *h, unsigned int size);
void deallocate(void *p);
```

Quando um bloco de dados com  $n\text{Bytes}$  é reservado com a operação *allocate()*, este tem duas partes:

- Metadados:
  - o Campo *available* que é um inteiro sem sinal (4 bytes). Pode ter o valor AVAILABLE (1) ou UNAVAILABLE (0).
  - o Campo *size* que é um inteiro sem sinal (4 bytes). Contém o tamanho em bytes da zona atribuída.
- Dados:  $n\text{Bytes}$  que o invocador da operação *alloc* pode usar como entender.

A figura seguinte apresenta um bloco com 100 bytes que está a ser utilizado.

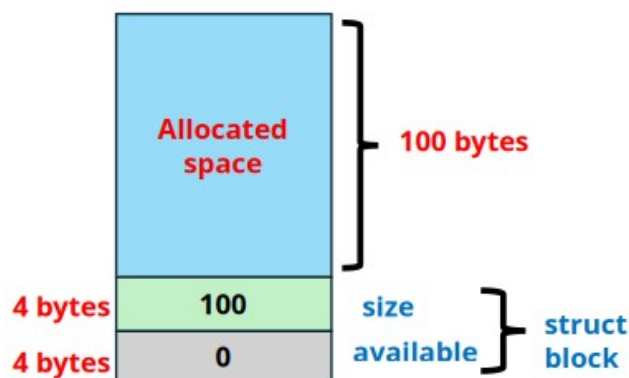


Figura 3 - Um bloco de memória com o seu descritor

O estado do espaço de memória é representado na estrutura *heap*, que contém os seguintes campos

- *base*: endereço inicial da área gerida
- *top*: endereço imediatamente superior ao do último byte atribuído por uma operação *allocate()*
- *limit*: endereço imediatamente superior ao do último byte da área gerida

A figura seguinte apresenta um exemplo de uma situação após a realização de várias operações *allocate()* e *deallocate()*.

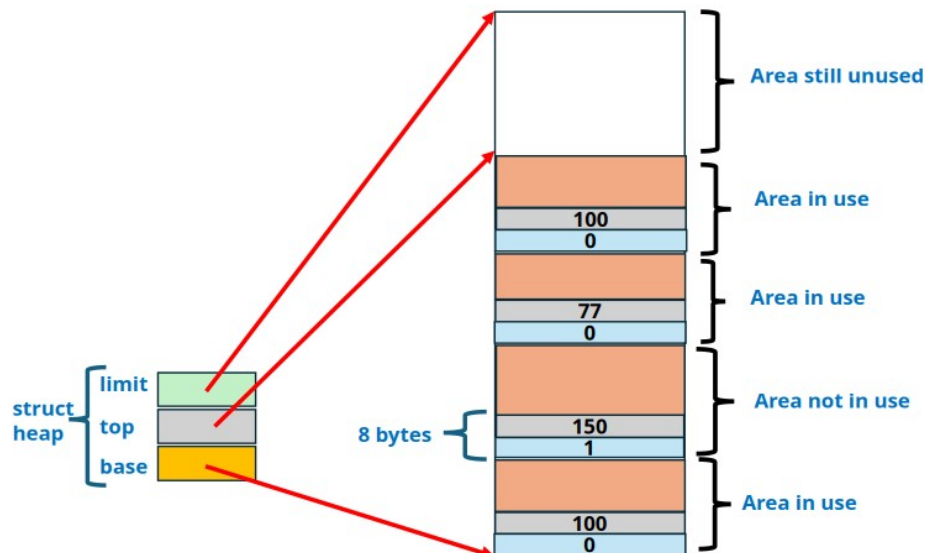


Figura 4 - Visão global

## Trabalho a realizar

Neste trabalho queremos implementar, em assembly `x86_64` as funções `allocate()` e `deallocate()` cujo comportamento se descreve a seguir.

**`void *allocate (heap *h, unsigned int bytesToAllocate);`**

Esta função devolve o endereço de um bloco com dimensão maior ou igual a `bytesToAllocate`. Para esse efeito percorre a memória descrita na figura 4, começando em `h->base` e saltando para a próxima estrutura `block`.

A função termina numa das duas seguintes situações:

1. é encontrada pelo menos uma estrutura `block b` que cumpre as seguintes condições:

`b->available == 1`

`b->size >= bytesToAllocate`

Se existir mais de um bloco nas condições acima, deve ser escolhido aquele que tiver o menor campo `size`.

Neste caso, retorna ao utilizador o endereço `b+sizeof(block)` e coloca `b->available` a 0.

2. se ultrapassa o valor `h->top`. Neste caso, deverá ser criado um novo bloco com tamanho `bytesToAllocate`. Em caso de sucesso, o procedimento será o mesmo do que em 1; a estrutura `h` deverá ser atualizada. Caso não seja possível criar o novo bloco porque se ultrapassaria o endereço `h->limit`, a função nada deve fazer e retornar -1.

**`void deallocate (void *address)`**

O endereço recebido é o bloco de dados (ver figura 3). A função terá de colocar no campo `available` da estrutura `block` a constante `AVAILABLE(1)`.

## Ficheiros disponíveis

Estão disponíveis no CLIP os seguintes ficheiros:

- o ficheiro `myAlloc.h` que define a API a usar

- um programa principal em C *main.c* completo. Este programa é invocado com dois argumentos:  
*./main sizeOfMemory maxBlock*

em que *sizeOfMemory* é o tamanho da memória gerida pelas funções *allocate()* e *deallocate()*. *maxBlock* é o tamanho máximo do bloco de memória usado nos testes realizados. O ficheiro inclui a invocação da chamada ao sistema *mmap()* que obtém a área de memória a gerir; a função *adjustToMultipleOfPageSize* ajusta *sizeOfMemory* para o menor múltiplo do tamanho da página usada pelo x86-64. A parte dos testes realizados pode naturalmente ser modificada.

- um ficheiro com um esqueleto do programa em assembly chamado *myAlloc.s*. Terá de completar este ficheiro escrevendo o código das funções *allocate()* e *deallocate()* com o comportamento descrito anteriormente.
- Makefile* que gera um ficheiro executável

```
CC=gcc
CFLAGS=-g -z noexecstack
ASFLAGS=-g -gstabs
all: main
main: main.c myAlloc.o
    $(CC) $(CFLAGS) -static -o main main.c myAlloc.o
myAlloc.o: myAlloc.s
    as $(ASFLAGS) -o myAlloc.o myAlloc.s
clean:
    rm -f *.o main
```

## Exemplos de resultados

Como exemplo, supondo uma implementação correta das funções *allocate()* e *deallocate()*, a execução do programa *main* deve produzir na saída um resultado semelhante ao seguinte, quando executada com “./main 8192 10” (note que os endereços podem variar):

Address	Status	Size
0x76900b9a9000	free	1
0x76900b9a9009	free	2
0x76900b9a9013	free	4
0x76900b9a901f	free	8
Address	Status	Size
0x76900b9a9000	busy	1
0x76900b9a9009	busy	2
0x76900b9a9013	free	4
0x76900b9a901f	busy	8
0x76900b9a902f	busy	10

## Modo de entrega

O ficheiro com as funções em assembly x86-64 deve ter o nome *myAlloc.s*. Esse ficheiro deve ser submetido ao Mooshak.